DuckFeed: Design Document
Columbia University, Spring 2011
CSEE 4840: Embedded Systems

Kevin Ramkishun Daniel Teger Julie Dinerman Scott Rogowski

March 22, 2011

1 Introduction

Duck Feed is a take on the classic game by Nintendo^{TM}, Duck Hunt. The game is played by a user yielding a "Zapper" gun originally created and popularized by Nintendo^{TM} in the 1980's. The user sees a varying number of Ducks appear to fly across the screen; points are accrued by successfully shooting (feeding) a Duck.

Duck Feed differs from Duck Hunt in a number of important aspects:

- Instead of being implemented for the Nintendo^{TM} Entertainment System, the game is implemented entirely on an Altera FPGA Board.
- The object of the game is to feed the ducks by shooting them with food, not to harm them!
- The game will be played on a CRT Computer Monitor, which has a different Refresh Rate than the original CRT Televisions of the 1980s (and thus requires modifications to hardware).

1.1 Milestones

1. Interface the Zapper hardware with the display to indicate when the trigger is pulled
2. Display simple Sprite-based animated graphics.
3. Have a bug-ridden game functioning

2 Overview

Duck Feed is implemented entirely on an Altera FPGA utilizing the Nios II microprocessor and custom hardware peripherals. The hardware is developed in VHDL and the software for the microcontroller is written in the C programming language. Custom registers provide the interface between the software and the hardware.

The hardware design consists of the following two Avalon Bus peripherals:

- The VGA Controller, which is tasked with drawing the stationary background for the game and rendering the Duck sprites on the screen.
- The Zapper Controller, which is responsible for interfacing with the Nintendo^{TM} Zapper: reading when the trigger is pressed as well as determining hits and misses.

The software design consists of the following modules:

- The Game Loop, which calculates the positions of the sprites on the screen and the score.
- The Zapper Interface, which communicates with the Zapper Controller hardware and lets the Game Loop know that a hit was made.
- The VGA Interface, which communicates with the VGA Controller hardware and which can set the visibility, position, and which rendering of a sprite to display on the screen.

The system is summarized by the Block Diagram in Figure 1.

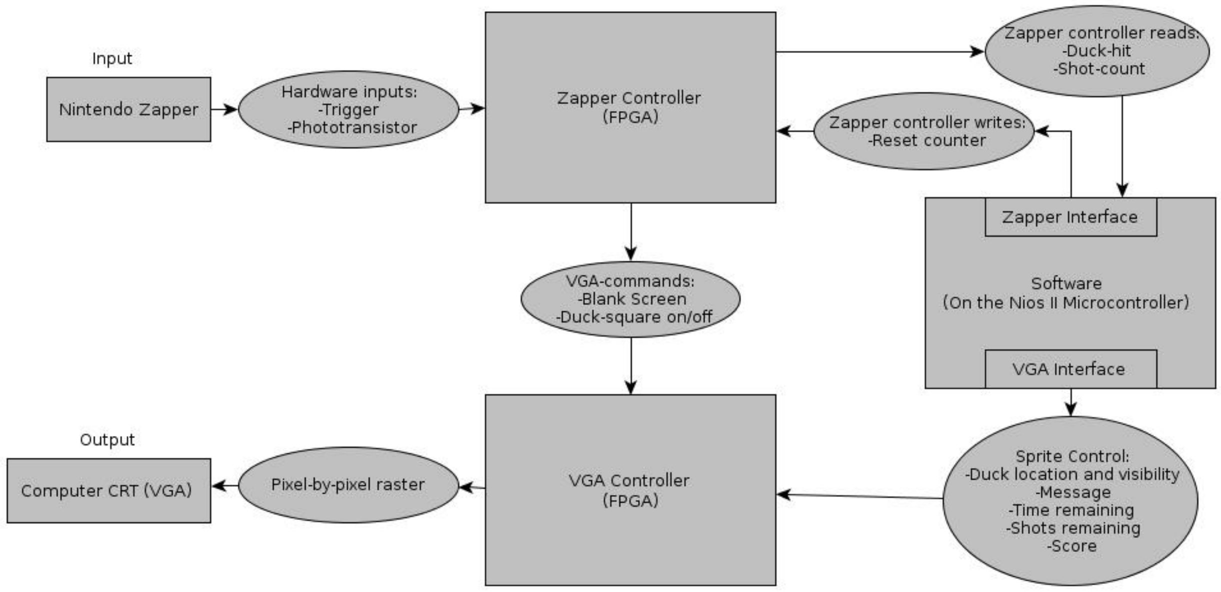
Input
Nintendo Zapper
Hardware inputs: -Trigger -Phototransistor
Zapper Controller (FPGA)
Zapper controller reads: -Duck-hit -Shot-count
Zapper controller writes: -Reset counter
Zapper Interface
Software (On the Nios II Microcontroller)
VGA Interface
VGA-commands: -Blank Screen -Duck-square on/off
Output
Computer CRT (VGA)
Pixel-by-pixel raster
VGA Controller (FPGA)
Sprite Control: -Duck location and visibility -Message -Time remaining -Shots remaining -Score

Figure 1: The Block Diagram of the Duck Feed system

3 Hardware Design

The hardware design includes custom Avalon Bus peripherals developed in VHDL and the Altera Quartus Tools, as well as modifications to the Nintendo^{TM} Zapper Light Gun. This section details all of the design decisions that are likely to be implemented in the final version of the Duck Feed game.

Note that some of the information in this section is not yet confirmed. This document shall be revised once we have the opportunity to take apart and analyze the Nintendo^{TM} Zapper. The information provided here is based on resources such as the original patent for the Zapper, as well as third-party articles and blogs dealing with customizing the Nintendo.

3.1 The Nintendo^{TM} Zapper Gun

Because the Zapper was conceived and developed in the 1980's, its operation utilizes very simple concepts. A "hit" or "miss" is determined by the following procedure, which occurs when the trigger is pulled:

1. The screen is made completely black for one frame.
2. The screen is kept completely black, except for the ducks, which are surrounded by a white box to distinguish them from the rest of the screen.
3. If the Zapper detected a completely dark frame, followed by a bright frame, it means that the gun was aimed at a duck, and therefore registers a 'hit'.
4. If the Zapper detected two completely dark frames, it means that the gun was not aimed at a duck, and registers a "miss".

If the Zapper did not detect one of the two above scenarios, then the Zapper was either not aimed at the screen or the user was attempting to cheat.

The above algorithm must also be extended for detection of multiple ducks. This is done using a binary search algorithm to minimize the number of screen flashes that occur. The algorithm is shown below:

1. Assuming there are n ducks on the screen, all n ducks are flashed.
2. If a hit was detected, then a binary search is performed as detailed below:
   (a) The ducks are split into two groups of $n/2$ ducks, named Group 1 and Group 2.
   (b) Group 1 is flashed. If there was a detect, then Group 2 is eliminated and we begin at Step 1 with Group 1 and $n/2$ ducks.
   (c) If there was not a detect, then we know the duck that was shot resides in Group 2. We begin again at Step 1 with Group 2 and $n/2$ ducks.

This algorithm above will complete in log2(n) frames where n is the number of ducks on the screen. Therefore, if the maximum 16 ducks are on the screen, the algorithm will take only 4 frames to complete.

The Zapper hardware is fairly simple. The detection circuit consists of a lens that focuses a light-detecting phototransistor and a band-pass filter. The trigger circuit is a switch connected to ground. The schematic of the circuit that was included with the original patent is shown in Figure 2.

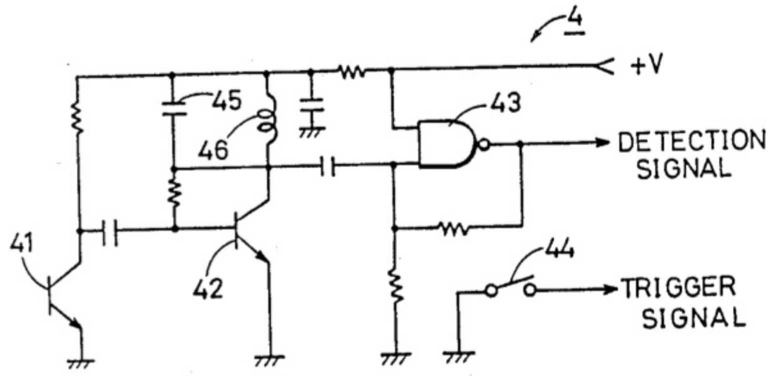

Figure 2: The schematic provided by the original Zapper patent

In the schematic, the capacitor and inductor labeled 45 and 46 comprise the band-pass filter. In more modern versions of the gun, this filter is replaced by an integrated circuit that performs the same task. The corner frequency of the filter is instead set by a resistor and capacitor. The filter bode plot included with the original patent is presented in Figure 3.

The band-pass filter in Figure 3 has a corner frequency at around $15kHz$, which *was* the horizontal refresh rate for CRTs in the 1980s. Because of this limitation, it was found through experimentation that the Zapper

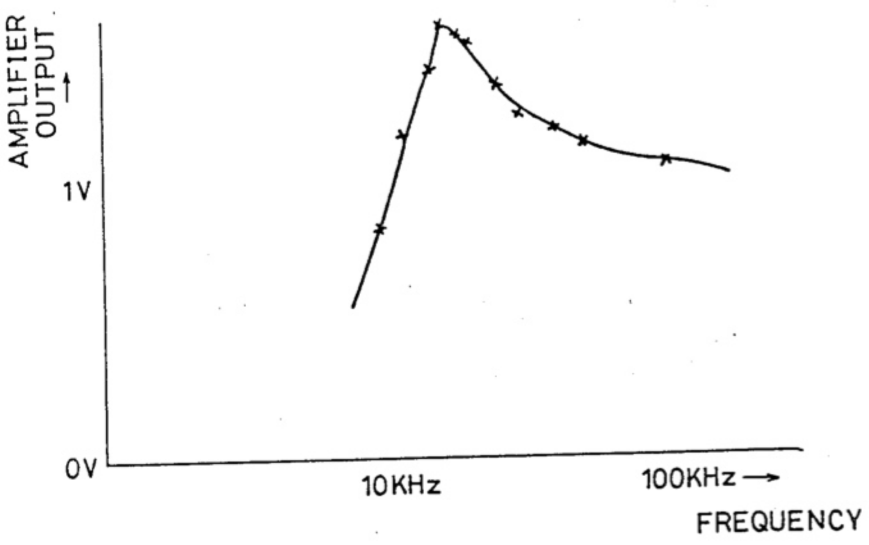

Figure 3: The filter Bode Plot provided by the original Zapper patent

fails to operate on a computer monitor CRT or LCD screen. Seeing as though the goal is for this game to operate on a computer monitor CRT, this filter must be altered to have a corner frequency near $31kHz$. This will likely be accomplished by altering the resistor value that sets the corner frequency by attaching a resistor in parallel with the one that is already in the gun. Unconfirmed sources state that adding a $390k\Omega$ resistor in parallel with the one that is already in the gun will accomplish this by creating an equivalent resistance of $180k\Omega$.

By altering the band-pass filter in the Zapper gun, it will operate with a CRT Computer Monitor. This allows us to use an existing monitor over VGA. VGA is chosen because we are familiar with the standard and may be able to reuse existing VHDL code.

3.1.1 Backup

In the case of unforeseen circumstances, a backup plan is also available for implementing the gun interface. This consists of building our own gun using a readily available photo-diode and a debounced switch. The simple circuit would be most likely mounted on a gun-shaped piece of cardboard. Again, this is merely a back-up plan in case the filter modifications do not make the Nintendo^{TM} Zapper usable for this project.

3.2 The VGA Controller

The VGA Controller will be based off of the de2_vga_raster.vhd file provided with the files in Lab 3. The input ports needed for the vga_raster include reset, a clock which is originally fed in as 50 MHz but will be divided down to 25 MHz for the VGA protocol, vertical and horizontal blank signals that tell the vga whether or not to draw the default background and vertical and horizontal sync signals that tell the vga_raster to draw something besides the default background.

The VGA Controller will render ducks and other images using the notion of Sprites. Since the background is a static image, the VGA Controller will always draw the background in hardware without any communication from the software. The software merely tells the VGA controller where and when to draw Sprites. This will be done using some hardware registers that are written to by software. Among these registers (detailed in the next section) are registers for the X and Y position of any one of the 16 ducks, registers for displaying the correct score, and registers for on-screen ASCII messages.

The controller will also have output ports VGA_R, VGA_G, and VGA_B that tell the controller which color to display in a each location on the VGA. The color output signals will be set according to logic that is performed in different processes in the architecture of the controller VHDL file. The controller will have front_porch and back_porch constants that will keep the controller drawing on the visible screen.

3.3 The Zapper Controller

The Zapper controller will be a VHDL construct that interacts with the Zapper's outputs. Its goal is to receive trigger pull signals and target detect signals from the gun and relay these messages to the software stored in the NIOS processor. When the trigger on the Zapper is pulled, the Zapper controller must enact the screen blank and target highlight mechanisms. This is accomplished by communicating directly with the VGA controller. First, the controller determines that the gun was pointed at the screen by blanking it. Then, the controller must establish that a duck was hit by highlighting each visible duck sprite with a white box. If a duck was hit, the controller must establish which duck was hit using the binary search algorithm detailed in section 3.1.

3.4 Other Hardware Considerations

This section highlights some miscellaneous hardware issues that will be considered in our design.

3.4.1 Level Shifting

The Nintendo^{TM} Zapper was created decades ago, and will undoubtedly operate on a different set of voltage levels than the Altera FPGA board to which it will be connected. Voltage level shifters will therefore have to be put into place to make all of the levels compatible to ensure safe operation.

3.4.2 Open Collector Inputs

The outputs on the Zapper (trigger and detect) are both open collector, so two pull-up resistors will need to be included on the FPGA board's input.

4 Hardware/Software Interfaces

The hardware and software communicate through a set of registers that are defined in the VGA controller and Zapper Controller. These registers are detailed in this section.

4.1 VGA Controller Register Definitions

Register Name	Format	Description
Duck sprite n x-position	16-bit unsigned int	Contains the X position (in pixels) of duck n.
Duck sprite n y-position	16-bit unsigned int	Contains the Y position (in pixels) of duck n.
Duck sprite visibility	16-bit binary	A binary digit for each duck determining whether or not that duck is visible.
Message	64-byte char string	Message to be displayed across the screen in ASCII. Useful for messages like "Game over!".
Shots remaining	16-bit unsigned int	The number of shots remaining used to determine how many bullet (food?) graphics to display on the bottom-left.
Time remaining	16-bit unsigned int	The amount of time remaining in 'bars' used to determine how many time 'bars' to display on the bottom-center.
Score	16-bit unsigned int	Contains the score which will be displayed in numerals on the bottom-right.

Table 1: The Registers for the VGA Controller. All of the following registers are written to by software and read by hardware.

4.2 Zapper Controller Register Definitions

Register Name	Format	Description
Zapper detect	16-bit binary	Hardware sets the binary digit corresponding to the index of the duck hit when the hardware logic has determined that the user has shot a duck. When the user has not shot a duck, all binary bits will be zero. Software then reads this register to determine the game logic.
Zapper shots-fired	16-bit unsigned int	Hardware increments this register when the user has pulled the trigger. Software will read this register to determine game logic.
Zapper reset shot-count	16-bit binary	Software sets this register to a specific value when a new level begins. Hardware reads this and will then reset its shot-count.

Table 2: The Registers for the Zapper Controller. Note that there will also be internal registers, not read or written to by software used to implement the target hit algorithm

5 Software Design

Software is written in the *C* programming language and compiled and debugged using the NIOS II Integrated Development Environment. The required software functions are listed in Table 3. Please note that these functions are not an extensive list of all of the functions in the software. We may decide to split functions that grow too large into a number of "helper" functions.

Name	Description
`void main()`	Begins the game by printing an introductory message and continuously calls each level function. If the user fails at any point, it prints a game over message. If all levels are completed, it prints a game win message.
`int level(int numDucks, int numShots)`	Begins a level with a given number of ducks and allowed shots. This consists of a game loop which continuously updates the screen and polls the hardware registers to determine if a shot occurred.
`int shotsFired()`	Reads the Zapper Control register to determine how many shots have been fired.
`int duckHit()`	Reads the Zapper Control register to determine if a duck has been hit.
`void resetCount()`	Resets the hardware shot counter.
`void printMessage(char *message)`	Prints a text message across the screen, such as "Game Over" or "Go!"
`void printTime(int timeleft)`	Updates the countdown graphic at the bottom of the screen.
`void printScore(int score)`	Updates the score on the screen
`void printShots(int shotsleft)`	Updates the number of shots left
`void printDuck(int duckIndex, int visible, int x, int y)`	Updates a single duck sprite

Table 3: A list of the software functions and their descriptions